009

009

Roy Weinstein
Chairman of the Board

Twenty-Fifth Floor
400 So. Hope Street
Los Angeles
California 90071
Tel 213 629 2655
Fax 213 688 8599
www.micronomics.com

MICRONOMICS
Economic Research & Consulting

May 13, 1998

P984238
B2375180009

Secretary
Federal Trade Commission
6th and Pennsylvania Avenue, N.W.
Washington, D.C. 20580

The enclosed discussion of potential Year 2000 problems is submitted in response to your request for comments.

Sincerely,

[illegible]

RW:shb
Enclosure

MICRONOMICS

Measuring Year 2000 Damages:

A General Approach to Estimating Losses Caused by Non-Compliant Software

by

James Ozenne
Atanu Saha
Roy Weinstein

February 1998

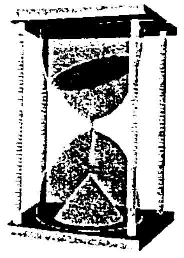
Measuring Year 2000 Damages:

A General Approach to Estimating Losses Caused by Non-Compliant Software

by

James Ozenne, Atanu Saha and Roy Weinstein[1]

A key finding of our analysis is that business disruption costs constitute only a fraction of the total damages associated with the Year 2000 Problem. The media and the existing research on this issue have largely ignored the other damage components: the time value of money and the benefits forgone in the case of a forced upgrade. We demonstrate that these damage components can be substantial and, under plausible assumptions, as high as the costs associated with business disruption.



Introduction

As the year 2000 approaches, MIS professionals around the world are busy identifying, repairing and replacing software affected by what has become known in the MIS world as the Year 2000 Problem. This problem, the use of two-digit fields to store the year of a date, has the potential to disrupt any business using the outdated software.

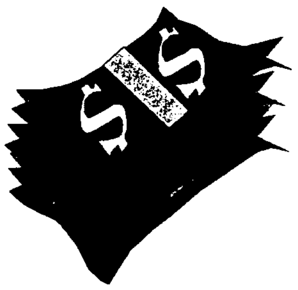
Although the Year 2000 Problem is familiar to the MIS community, it remains less well understood by the rest of the business world. Essentially, the problem arises from the use of two-digit year fields, a practice intended to economize on computer resources. For instance, the year 1998 is stored as "98" rather than "1998". Under this scheme, the year 2000 is stored as "00", with no way for the computer program to distinguish between the years 2000 and 1900. The practice of writing programs using this date coding developed in past decades, when software was written under the assumption that it would no longer be in use by the time the year 2000 arrived. However, as this software has developed incrementally into today's systems, many two-digit date fields remain. When a post-1999 date is entered into one of these systems, the program may treat the record as if it were 100 years old, triggering inappropriate action of some kind. Such records might be deleted, ignored, or sorted incorrectly, or might cause the program to crash or produce nonsensical error messages or other unexpected results, depending on the nature of the software.

[1] The authors are employed at Micronomics, Inc., an economic research and consulting firm in Los Angeles and Washington, D.C. They are grateful to Andrew Steinberg for his assistance.

The question of which computer systems will be affected by the problem, and to what extent their failure will disrupt business, continues to be debated. Most attention, though, seems to be focused on large mainframe applications. Many of these programs are based on old code written in COBOL, a business programming language widely used in the 60s, 70s, and 80s. Companies now using the doomed software are being forced to take action. Some will elect to repair their existing systems, perhaps by hiring COBOL programmers to comb through their software and change two-digit year fields to four-digit ones. Other companies will replace their systems entirely with state-of-the-art software, thereby avoiding Year 2000 problems and also gaining additional functionality. Companies facing expensive software repairs are eager to mitigate these expenses, not only by implementing cost-effective software solutions, but also in some cases by holding the sellers of the defective software financially responsible.

In order to calculate the amount of damage in these cases, we must consider the direct cost of installing new software and any business interruption that may result from Year 2000-related software failures. In cases where a company uses the Year 2000 as an opportunity to install software that is not only Year 2000 compliant, but also provides the company with significant new benefits, we must consider the change in the company's position with respect to its software upgrade path.

In this paper, we present a simple model for estimating damages related to the Year 2000 Problem. Without attempting to incorporate the complexities of the software market or any specific technology, we take a general approach that might be used to organize and evaluate the details of any Year 2000 software damages case. A key finding of our analysis is that business disruption costs constitute only a fraction of the total damages associated with the Year 2000 Problem. The media and the existing research on this issue have largely ignored the other damage components: the time value of money and the benefits forgone in the case of a forced upgrade. We demonstrate that these damage components can be substantial and, under plausible assumptions, as high as the costs associated with business disruption.

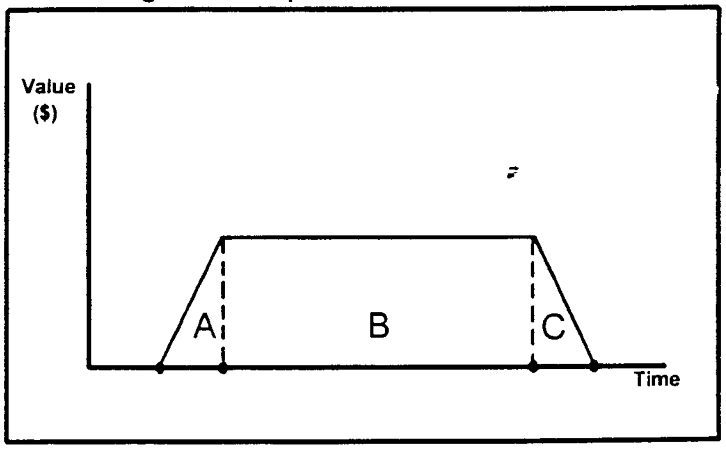


The Value of the Software Asset

Firms invest in software because they expect that it will provide some benefit in the future. Most software has a useful life of several years and is installed with the intention that it will be used on a continual basis throughout its life. The value added by a software system is the sum of the stream of value it generates over its useful life. Effective use of software may improve worker productivity, resulting in greater output capacity and reduced payroll. Many firms use software to manage and optimize their supply, production, sales, and delivery processes. The net value of a piece of software to a company depends on its ability to provide these benefits over time, weighed against the costs associated with software installation and operation, such as system maintenance, user training, and technical support.

We can visualize the stream of value generated by a piece of software using a graph as in Figure 1. In this graph, the vertical axis represents the software's value. Specifically we can think of this quantity as dollars per day of increased profitability for the firm. The horizontal axis represents time. The area under the entire curve represents the total value provided by software over its useful life.

Figure 1: Simple Software Value Stream



In this diagram, we have shown three distinct regions representing a life cycle that might be associated with any piece of software. The upward sloping region at the beginning of the cycle, labeled area A, could be explained by a gradual implementation where not all the functionality is available at once. A user training period, where the operators first learn the basic features and then become more sophisticated users, is another possible explanation.

The flat region of the curve, area B, is a period of constant production from the software system. This is probably a good approximation for any program that has been in day-to-day use for some time, especially if the business and technology are fairly stable.

Area C, the downward sloping region, represents a decline in the software's usefulness. Such a decline might occur if the software is designed for hardware that becomes obsolete and expensive to maintain. Another important possibility is that changes in technology or the marketplace make the software less valuable.

Although the software might still perform as well as ever, the work it is designed to do no longer suits the needs of the company. Area C might also represent a period during which the software is being replaced -- that is, the software's functions are gradually switched over to a new system.

It may be problematic to accurately measure a software system's value at any given point in time. However, most companies, in making effective decisions about which software to choose, confront the problem by attempting to determine the relative value of various software systems. Measuring such an intangible quantity has no single best method, and companies indeed use a range of approaches from the gut feelings of managers to large-scale, comparative benchmarking processes and detailed cost-benefit studies. There are also research firms that collect survey data and help companies to value and compare software packages. These sources may not render complete information about the value of software, but certainly provide some starting point.

Investing in Software

To begin reaping the benefits of any software system, a firm must first make some initial investment. This investment includes the purchase of finished software or the cost of programming new software, as well as the cost of user training, installation, and any interruption in the business that might result from the installation process. Companies weigh these costs carefully against the software's benefits.

Installation costs, unlike the software's value stream, are incurred over a relatively short period of time. Many of the initial cost components are directly measurable: the price of a software licensing fee, consulting fees and salaries for programming, customization, and installation work, and the cost of user training sessions. Other components such as down time and lost productivity during the installation period may not be directly measurable but are still tangible costs that many firms are able to quantify.

It is important to note that the installation cost of software is often quite considerable. SAP systems, which are a popular choice for companies seeking to replace their non-compliant systems, can cost more than four times the software license fee to install. Bruce Richardson, the Vice President of Research Strategy for Advanced Manufacturing Research, says, "our research shows that for every dollar spent on SAP, at least five dollars are spent on hardware, databases and tools, consulting and services, networking and complimentary applications."[2] Firm owners are quite aware of these costs as well as the lost productivity that occurs when new software is installed.

Another important factor for any company repairing or replacing software during the near-2000 time period is the current market for programming services. The scramble to fix the Year 2000 Problem has created a seller's market for programmers, especially those who can work in languages, like COBOL, that are important to the Year 2000 rush. According to John A. Bace, Research Director of Gartner Group, "In 1998, almost anyone with those skills will be able to find jobs at nearly a 50% premium over 1996 salaries."[3]

[2] Cafasso, Rosemary, "AMR Launches the SAP Advisory Program," PR Newswire, 8/22/97, p822NEF005.

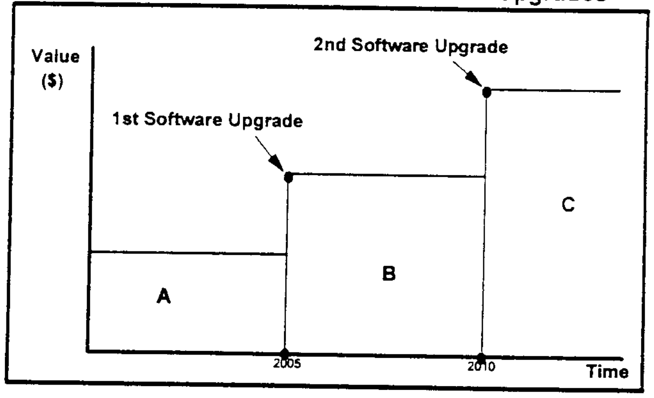
[3] Gross, Neil, Amy Cortese and Steve Hamm, "Software," Business Week, 1/12/98, p. 86.

The Upgrade Process

Many factors -- a software program's increasing or decreasing value over time, the relative values of old and proposed new software, upgrade installation costs, and changes in the business and technology -- are important to a firm's choice and timing of an upgrade. The rapid advance of technology often forces firms to begin addressing the issue of software upgrades very early on -- sometimes even before their original systems are installed. In deciding to upgrade, a firm must consider not only the costs and benefits of the new and improved software, but also the forgone benefits of the replaced system. The firm may want to dump old software that does not work on new hardware, or may find that the old system cannot accommodate new business requirements. On the other hand, the users' proficiency with the old software may be so well developed that a company will be reluctant to make a change. The promise of a superior system becoming available in the future also might tempt a company to delay upgrading.

In Figure 2, we present the software value stream for a company as it goes through an upgrade. Note that for simplicity, we have dispensed with the ramp-up and declining phases of the software's life cycle and left only the constant region, keeping in mind that real-world software is likely to increase and decrease in value over its life. Also, note that the new software provides a higher level of value than the old. No firm will replace its software unless it expects the new product to perform better than the old one could have. Therefore, the constant value assumption in this diagram requires that the value stream of the upgrade be higher than the original.

Figure 2: Value Stream of Successive Upgrades



Uncertainty plays a crucial role in the upgrade decision. A company must make some assumption about the future of the business, as well as the future of available software technology, before implementing a particular software upgrade. Technological advancements, industry downturn, and changes in business requirements are uncertain elements that make companies cautious when deciding on a new software system. Once a company commits to a new system, reversal can involve substantial costs. Companies frequently delay upgrade decisions for this reason. Uncertainty as to whether new software will perform as expected also encourages caution. In an uncertain world, a firm's flexibility is an intangible but important asset.

Figure 3: Software Upgrade Productivity Comparison

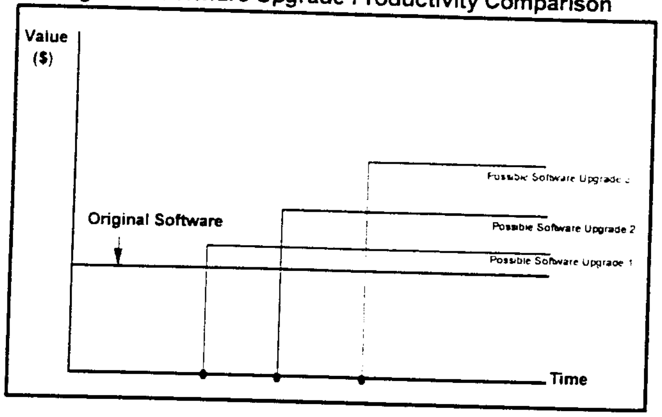
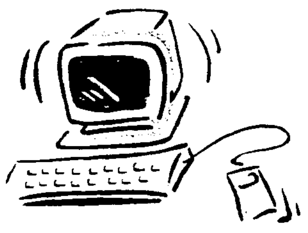


Figure 3 demonstrates one aspect of this flexibility issue. In this diagram, a firm faces the decision of upgrading or delaying the decision. Delaying may mean that new software technology will arrive, or that future knowledge of the industry will provide better guidance in the choice of software.



Impact of the Year 2000

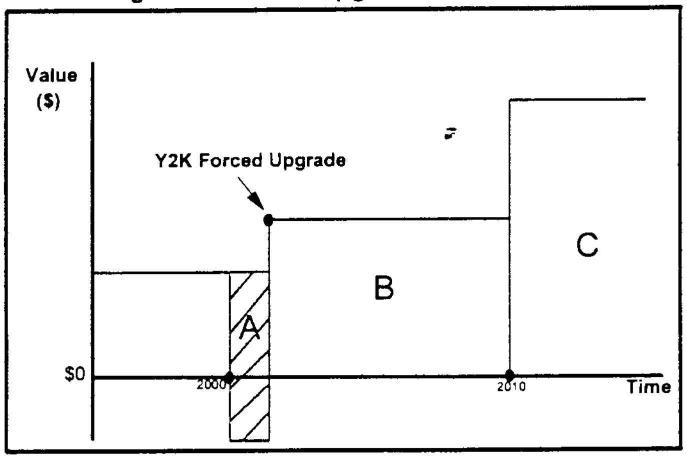
Companies now using non-compliant software find themselves in a difficult situation. First of all, many firms are uncertain of the extent to which the Year 2000 Problem will affect their software systems. It also is unclear how costly Year 2000 software failures will be to any business. In this uncertain environment, different firms are pursuing different paths in their bid to survive the crisis. Whether firms replace many of their systems, or hire programmers to repair their date fields, or even delay action in hopes that software failures will not significantly impact them, these firms experience two distinct damage components: the cost of any business interruption caused by a software failure, and the cost of being forced to repair or replace non-compliant software in order to avoid business interruption.

Business interruption costs associated with Year 2000 failures are conceptually simple. A software failure might cause a firm to lose, delay, or mix up orders or invoices, resulting in decreased revenue and angry customers. Inventory and shipping mistakes are likely to be costly, as are production shutdowns that could result from a severe software failure. These costs will occur largely in the year 2000, but in many cases have begun already and will continue into the new millenium. Estimating the costs of a Year 2000 software failure is by no means easy, but is certainly possible. Insurance companies that provide business interruption coverage perform this type of calculation regularly. Of course, many firms will be fully compliant by the time the new millenium arrives, thus avoiding any business interruption costs.

The second component of Year 2000 damage, the cost of repairing or replacing non-compliant software, is much more complex. The complexity results from a firm's option either to replace the software with a system that has greater functionality or to use the date field repair project as an opportunity to build greater value into an existing system. Once a firm has responded to the Year 2000 crisis by installing software that is superior to its pre-2000 systems, it begins to benefit from a higher level of software value. It also has suffered the flexibility loss discussed in the previous section. In fact, the firm has embarked on a completely different upgrade schedule from one that it may have been planning before learning of the Year 2000 Problem.

Figure 4 illustrates the situation of a firm faced with a Year 2000 software problem. In this diagram, the firm experiences a period of business interruption, beginning at the year 2000, in which the defective software actually hurts the business. The benefit curve lies below zero in this situation, and the resulting diagonally shaded region represents the direct costs associated with the software failure. Once the firm has successfully installed a replacement system, it enters area B, a period of positive software benefits.

Figure 4: Forced Upgrade Value Stream



This firm may have preferred to delay any upgrade decision, but the software defect forces it into action. As a result, the firm may miss out on advances in software technology that occur towards the beginning of the new millenium. Although the firm begins benefiting from the software upgrade right away, this solution may prove more costly than necessary if business requirements change and the company is soon forced to upgrade again, or if an industry downturn makes it impossible for the company to recoup its investment.

The fact that the firm incurs the entire installation cost of a software upgrade shortly after the year 2000, rather than delaying this cost, results in a loss of time value of money. The firm also pays the Year 2000 programmer premium for this upgrade because it is forced to act during a period of high programmer compensation. Additional costs may result from the firm's need to rush the installation process, rather than taking it at a slower, more efficient pace.

Adding Up Damages

In order to systematically organize a firm's Year 2000 damages, we compare the real world, in which the firm is forced to repair or replace some non-compliant software and experiences some software failures, with an alternative world in which the firm's software is Year 2000 compliant and can be used without failure into the new millenium.

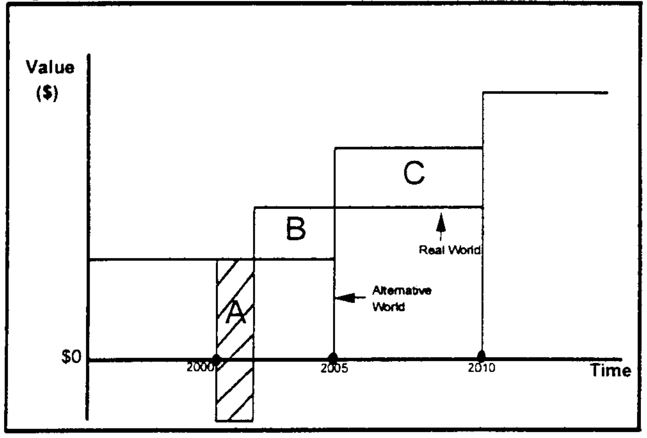
For example, suppose a fictitious firm, XYZ Wholesalers, Inc., uses an aging software system to keep track of orders, shipping and inventory. Unaware that this system is riddled with two-digit date fields, the company enters the year 2000 completely unprepared. As the new year begins, parts of the system fail, forcing workers to abandon many of the system's features and keep track of shipping schedules by hand. The company is forced to hire additional administrative workers to compensate.

XYZ immediately hires a consulting firm to install a completely new, more powerful piece of software, a move the company had anticipated making in 2005. The system takes one year to install and costs a total of $5 million. During that year, the expense and lost business due to the old software's failure amount to an additional $2 million loss.

For this example, we assume that the new software is functionally superior to the old software. Its value to the company is $500,000 per year more than the old system's was. However, the year 2000 software is not as sophisticated as the anticipated year 2005 upgrade would have been; its value falls $500,000 per year short of that level.

In Figure 5, we compare the real world of XYZ, shown in black lines, with an alternative world in which XYZ is able to proceed with its plan of upgrading its software in 2005, shown in red. Note that in both the alternative world and the real world, XYZ upgrades again at the year 2010. This assumption means that all of the damages are incurred before that point, and allows for a simple diagram and simple calculations. We could certainly create a much more complex scenario. It is probably not unrealistic, however, to assume that the real and alternative worlds converge at some point, given that upgrades may be motivated by external factors such as a major advance in software technology, a new business requirement, or a company-wide reorganization.

Figure 5: Real World vs. Alternative World Comparison



The damage calculation begins with the cost of the Year 2000 software failure, represented by the diagonally shaded area A. This is the $2 million in expenses and lost revenue caused by the failure of the old software.

The next damage component is the $5 million setup cost of the new system. However, in this case, XYZ has avoided perhaps a $5 million setup cost in the year 2005, the year it originally planned to implement a major software upgrade. Assuming a six percent discount rate, the cost to the firm of upgrading in 2000 rather than 2005 is $1.26 million.

At the same time, we must consider the added value of the software purchased in 2000 and that of the forgone upgrade in 2005. In 2001 through 2004, the firm derives more value from its new software than the old software would have provided. The present value of this $500,000 per year gain is $1.73 million, represented by area B on the diagram. However, in 2005 through 2009, XYZ misses out on the added value of the year 2005 technology, which is $500,000 per year better than the year 2000 software. The present value of this loss is $1.67 million, represented by area C. Subtracting area C from area B, we have a net gain of $0.06 million due to the altered value stream of the software.

The total damages experienced by XYZ due to the Year 2000 Problem can now be calculated as:

area A + setup 2000 - PV(setup 2005) - PV(area B) + PV(area C)

Where area A is the failure cost of $2 million, setup 2000 is the total software installation cost in the real world ($5 million), PV(setup 2005) is the present value of the software installation cost in the alternative world ($3.74 million), PV(area B) is the present value of the software value stream improvement in the real world ($1.73 million), and PV(area C) is the present value of the software value stream improvement forgone in the real world ($1.67 million). Substituting the numbers in our example gives us:

$2 million + $5 million - $3.74 million - $1.73 million + $1.67 million = $3.20 million

Thus, the damages comprise three components:

1. Business disruption costs	$2.00 mm
2. Time value of money	$1.26 mm
3. Altered value stream of software	-$0.06 mm
Total	$3.20 mm

Two important general conclusions follow from the analysis. First, business disruption costs, which have captured most of the media attention, make up only a fraction of the total damages suffered by the firm. Other components can be substantial. We have not quantified the damages arising from the loss of flexibility discussed earlier, which may add significantly to the total.

The second general observation is that the timing of the alternative and real world upgrades substantially affect the damage estimate. The later the company planned on upgrading before discovering the Year 2000 Problem (the start of area C), the greater the damages. This is because the more the firm is forced to diverge from its plans, the greater the loss of time value of money from the software setup costs. The firm's loss of flexibility also follows this pattern, since the early upgrade robs the firm of the benefit of information it would have applied to the future upgrade decision. Similarly, the software's longevity is important: if the upgrade at the end of area C were to occur in 2015 rather than 2010, area C would be larger and would represent a loss of $2.91 million. The altered value stream of the software, area C minus area B, would then result in a net loss of $1.18 million, increasing our total damage estimate to $4.44 million.

In Review

In this paper, we have represented software as an asset that derives its value from a stream of benefit it provides to a firm. Furthermore, we have assumed that firms periodically replace their software, incurring some costs of installation and training. These costs are recouped over time, as the company reaps the benefits of newer, more productive software.

A company that unexpectedly faces a software failure due to the Year 2000 Problem may incur business disruption costs. However, we have shown that the firm is likely to suffer from substantial damages unrelated to business disruption. The Year 2000 Problem forces a firm to deviate from its planned software upgrade schedule. It is forced to incur upgrade costs sooner than anticipated, resulting in a loss of time value of money. Also, because of the forced upgrade, the firm misses out on benefits from advances in software technology that occur at the beginning of the new millenium. Our analysis shows that the loss associated with these costs can be substantial and may be as high as those associated with business disruption.

It is undoubtedly true that every company's software upgrade profile is unique and so is the damage associated with its Year 2000 problems. We believe our model provides a general framework and identifies the features common to most companies' problems. It is, thus, a reasonable starting point for the analysis of damages in any particular case.

Micronomics, Inc. is an economic research and consulting firm with offices in Los Angeles and Washington, D.C. Our staff of economists, econometricians, statisticians and computer professionals serves a national and international client base. We specialize in the application of price theory and economic analysis to real-world problems that require practical, yet sound, solutions. Our work focuses on merger analysis, other antitrust issues, the valuation of intellectual property and regulation. We also have extensive experience in areas such as damage assessment and securities fraud. Our clients include law firms, publicly and privately held businesses and government agencies.

For more information, please contact Roy Weinstein, Chairman at (213) 629-2655.

Micronomics, Inc.
400 South Hope Street
Suite 2500
Los Angeles, CA 90071
Tel: 213 629-2655
Fax: 213-688-8899
www.micronomics.com

Micronomics, Inc.
1201 New York Avenue
Suite 530
Washington, D.C. 20005
Tel: 202 408-0272
Fax: 202-408-0273
www.micronomics.com